RESEARCH STATEMENT

Lin, Yun (llmhyy@gmail.com)

My research aims to develop practical tools to facilitate the efficiency of programmers. I have developed a series of tools (manifested as Eclipse plugins) to facilitate programmers to utilize code duplication for practical use [3, 8], summarize code pattern for code reusability [1, 7], keep code consistent to its design [2], and localize bugs more efficiently [5, 6].

During my Ph.D. study (2012–2015), I worked on code clone [1, 3, 7, 8] and code refactoring [2]. Code clone is a phenomenon where duplicated code is distributed among the whole software system. Research literatures show that 5%–30% of the code are duplicated in industry. Different from traditional research on how to detect code clone as "bad smell", my insight lies in that code clone also implies undocumented code pattern that we can follow. Therefore, my Ph.D. research focuses on how to build a platform to utilize code clone to facilitate software development and maintenance. To this end, I first proposed a clone-differencing algorithm to efficiently detect the difference of multiple duplicated code [8]. Based on this work, I further propose an approach to suggesting where and how to modify programmers' pasted code [3] and an approach to identifying undocumented recurring designs in the system and summarizing them into reusable templates for code generation [1]. In the same line of software maintenance, I proposed an approach to refactoring code into a user-specific design [2]. Noteworthy, all these works are supported by solid tools in the form of Eclipse plugin, their source code is available on my Github website (https://github.com/llmhyy).

During my postdoctoral training (2015–now), I begin to use static and dynamic program analysis technique to automate program debugging and testing. One big challenge for automating debugging is lack of oracle, i.e., the "correct version" of program usually exists only in programmer's mind. To tackle this problem, my ICSE'17 paper describes a feedback-based debugging approach [1], which asks for programmers' feedback as partial code oracle and interactively recommends suspicious steps on the buggy execution trace. Moreover, the feedback-based debugger can estimate programmers' debugging decision after the programmer has provide several his or her debugging feedback. In addition, I further propose a data-driven approach to improving the debugging facility of traditional dynamic slicing in my ASE'18 work. Based on the trace collection technique in above work, I collaborate with Prof. Liu, Yang at Nanyang Technological University to build a regression fault localization technique, which can automatically compare a correct trace and a buggy trace to generate explanation for a regression bug [10]. Based on the above two works, I am developing a simulated debugging approach to imitate programmers' debugging behaviors by checking data and control flow of buggy program [4]. By collecting a large number of simulated debugging result, I am able to feed the debugging data into a deep learning model for more accurate recommendation for fault localization [5] (published in ASE'18), overcoming the limit of traditional program analysis techniques. Moreover, I am cooperating with Prof. Sun, Jun in SUTD to leverage machine learning approach to improve traditional test coverage [9].

In this research statement, I first retrospect my previous research. Then I summarize my research philosophy, before looking ahead my future research directions and working plans.

Previous Research Contributions

Utilizing Clone-Based Pattern for Software Maintenance

Code clone is a phenomenon where code duplication is distributed among the system, it is very prevalent among industry software systems. Many software engineering researchers have devoted themselves on this topic in terms of its detection, evolution, removal, and management. Different from all existing approaches, my research on code clone lies in how to utilize code clone for supporting practical software development and maintenance. To this end, my research starts with clone difference analysis [8]. Based on how duplicated code are different from one another, I extract undocumented rules of program convention to help programmers avoid potential fault [3] and save programming effort [1].

Clone Differencing Algorithm. In year of 2014, I proposed a clone differencing algorithm for analysing the difference of multiple pieces of cloned code fragment in an efficient way [8] (in ICSE'14) and enhance this technique by taking code syntactic boundary into consideration [3] (in FSE'15). The key challenges lie in (1) tackling the trade-off between the complexity and precision of matching the global code sequences and (2) distinguishing same code tokens by their different syntactic features during code matching. I address the first challenge with customized progressive alignment procedure, and the second challenge by defining metrics for evaluating tree distance for different AST nodes. The tools are published at my Github website (https://github.com/llmhyy/mcidiff).

Recommendation for Modifying Pasted Code. Based on the support of differencing technique in my ICSE'14 paper, I further propose an approach to recommending where and how to modify a piece of pasted code [3]. The rationale is that, I regard all the duplicated code as historical copy-and-paste results, and their differences as the historical

modifications on the pasted code. Therefore, the pattern of clone difference can be learned to suggest how to modify a new pasted code. The tools are published at my Github website (https://github.com/llmhyy/ccdeamon).

Extracting Recurring Designs for Code Generation Template. Still based on the support of differencing technique, I proposed an approach to identifying recurring design in software systems and the identified recurring designs can be automatically extracted into design templates for code generation [1]. I observe that programmers sometimes copy and paste related pieces of code. An keen observation is that they are copying more a design than several pieces code. Therefore, similar designs recur many times in the software system. Recurred designs indicate undocumented project-specific convention or knowledge. Therefore, I developed a technique to automatically identify recurring designs in the system. In addition, these identified recurring designs will be extracted into code templates manifested as UML class diagrams to generate code skeleton as well as body. A video of the tool is available at http://linyun.info/micode/index.html and the source code is available is at https://github.com/llmhyy/MICoDe.

Interactively Aligning Code and Design with Search-Based Technique

In the same line of software maintenance, I also developed a technique to keep the consistency of code and design [2]. With the evolving of the software development, the code is usually deviated from its original design. In order to avoid such deviation, programmers usually need to refactor code manually, which is a time and effort consuming task. Therefore, I developed a technique called Refactoring Navigator to automate the consistency of code design and its implementation.

Refactoring Navigator regards user-specific design as target and code implementation as source, and automatically generates a refactoring solution consisting of a sequence of refactoring steps leading the source to the target. Regarding programmers have their own preference on how to refactor their code, the technique allows them to accept or reject certain refactoring steps as user feedback. Based on these feedbacks, Refactoring Navigator will generate a new sequence of refactoring steps. Refactoring Navigator is applied on both student code assignment and an industrial system to show its effectiveness. Its source code is available at https://github.com/llmhyy/Refactoring-Navigator.

Feedback-based Debugging and Learning-based Testing

During my postdoctoral training, I transfer my interest to automate program debugging and testing.

Feedback-based Debugging. One big challenge for automating debugging is that the "correct version" of program usually exists only in programmer's mind. To tackle this problem, my ICSE'17 paper describes a feedback-based debugging approach [1], which asks for programmers feedback as partial code specification and interactively recommends suspicious steps on the buggy execution trace. Based on the trace collection technique in this work, I collaborate with Dr. Wang, Haijun and Prof. Liu, Yang at NTU, to build a regression fault localization technique, which can automatically compare a correct trace and a buggy trace to generate explanation for a regression bug [10]. The source code is available at https://github.com/llmhyy/microbat and https://github.com/llmhyy/tregression.

Learning-based Debugging. On tackling the debugging problem, I observe the limit of dynamic slicing, a traditional approach to locating relevant program statements based on a given statement. That is, dynamic slicing comes into a dead end when a software bug is caused by missing some code or missing the execution of some code. To this end, I proposed a data-driven approach to enhance dynamic slicing by building a neural network to predict the location where code or the execution of code is missing [5].

Learning-based Testing. I am also working on a software testing project which aims to leverage machine learning approach to improve traditional test coverage [9]. The idea is to learn a linear model from test inputs to approximate which inputs can exercise the true or false branch of complicated branch node in CFG. The linear property of the model can further allow us to solve a test input so that it can reach a specific branch. This work is under the review of TSE [9].

Research Philosophy

My research philosophy is as follows:

Tackling Important Problems. I pursue to address important problems to serve more people. Software engineering is usually considered to serve the benefits of software engineers. In my research, I would like to extend the application of software engineering techniques, such as dynamic and static program analysis, to areas such as security and HCI so that my research can gain more impact and serve more people in the world.

Delivering Practical Tools. Each of my previous research work is supported by an open source Eclipse-plugin project. I would like to deliver solid research prototypes for the benefits of both research community and future industrial collaborators. First, I aim to build toolset or platform that the researchers in the same community will find useful to develop their own research. I am now building a flexible trace collection agent, which can be widely applied to facilitate dynamic analysis of Java program (https://github.com/llmhyy/microbat). Second, I also aim to deliver research prototypes to inspire or help industry people to build their own practical tool.

Building Strong Collaboration. Good research idea can be sparked by discussing with different people from different fields. Moreover, good research gains its impact by being applied in industry and many experimental ideas should withstand the test of industrial application. Therefore, I will build more collaboration with people from both academia and industry.

Future Research Directions

I will establish my future research direction by expanding my software engineering research (software debugging/testing and code recommendation) to serving more people than software engineers. With my experience of analysing and manipulating Java bytecode and AST, I will proceed to three directions of research, i.e., code performance optimization, security analysis, and AI debugging and testing.

Code Performance Optimization. Performance bugs are software faults where software performance fails to meet the expectation. With my experience of code instrumentation, I can extend my pattern mining work from static code to dynamic execution trace. Up till now, trace-based approaches are usually applied for identifying functional bug. Nevertheless, recurring patterns on a trace are an important signal indicating unnecessary computation. Based on instrumentation technique and pattern mining on traces, it is natural to propose approaches to trading algorithm space for time, or vice versa.

Security Analysis. It is natural to extend software debugging and testing techniques to security, such as identifying software vulnerabilities and diagnose their root causes to generate potential patch and search exploitation. I am now working on a software testing project which aims to improve testing coverage by active learning [9]. Exercising all the branches of the control flow graph (CFG) of a program is notoriously challenging. The idea is to (1) actively learn a linear model from some test inputs to distinguish test inputs exercising true and false branch of some conditional nodes on CFG and (2) generate the test inputs based on learned linear model to increase the probability of covering a specific branch. The experimental results have shown that our approach is more effective than both concolic testing and random testing. The same technique can be applied in fuzzing binary programs. Moreover, we can complement this technique with existing search-based meta-heuristic algorithm to detect software vulnerabilities in a more efficient way.

AI Debugging and Testing. The world is investing on AI projects dramatically, which will incur a soar on the number of AI developers. Therefore, how to facilitate debugging and testing on AI program will be a great and important challenge.

Different from debugging and testing traditional program, AI program such as deep learning techniques usually aggregates inputs from a large corpus of training data and its behaviours are usually nondeterministic. In this light, the traditional program analysis techniques such as data or control flow analysis probably do not work anymore. Traditional debugging cares about how to diagnose the causality chain from the revealed fault to its root cause. The chain is usually fixed and deterministic. However, the "chains" in AI programs are usually overwhelming. Based on my experience of summarization and visualization technique, we can tackle this problem in a different way. We can apply visualization technique for AI debugging to help developers localize the fault from a big picture instead of struggling with low-level details.

On the other hand, testing AI program to ensure its reliability is also a very important topic. For some popular AI techniques such as deep learning, an important question is how to actively generate test inputs of an AI program so that these test inputs can be located near the classification boundary. However, the boundaries of AI program such as deep learning model are usually hard to describe or interpret. During my postdoctoral training period, we are starting a project joint with Huawei, Singapore, for AI testing. We are using both first and second gradient-based techniques to explore the boundary. Based on such a technique, we aim to generate some counter examples of a learning model to reveal its vulnerabilities.

Collaborative Research

It is important to build reliable collaborative research to be a successful faculty. In the past few years, I established collaboration with Prof. Xing, Zhenchang (who was an Assistant Professor in NTU and a Senior Lecture in ANU), Prof. Cai, Yuanfang (who is an Associate Professor in Drexcel University, USA), Prof. Sun, Jun (who is an Associate Professor in SUTD), Prof. Liu, Yang (who is an Associate Professor in NTU), and Prof. Yang, Zijiang (who is a Professor in Western Michigan University, USA). Now, I am strengthening my connection with these excellent researchers and building more connection during my attending at different top-tier software engineering conference. Moreover, during my Ph.D. study and postdoctoral training, I also help collaborate with research lab in Huawei (in Shenzhen and Singapore).

In summary, I am looking forward to pursuing an academic career in your prestigious university. With my research ability and the world-class resources in your university, I believe I can build a strong research group and grow up to be an influential scholar in the program analysis and software engineering in the next 3-5 years.

References

[1] Yun Lin, Guozhu Meng, Yinxing Xue, Zhenchang Xing, Jun Sun, Xin Peng, Yang Liu, Wenyun Zhao, and Jinsong Dong. Mining implicit design templates for actionable code reuse. In *Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering*, ASE 2017, pages 394–404, Piscataway, NJ, USA, 2017. IEEE Press.

[2] Yun Lin, Xin Peng, Yuanfang Cai, Danny Dig, Diwen Zheng, and Wenyun Zhao. Interactive and guided architectural refactoring with search-based recommendation. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2016, pages 535–546, New York, NY, USA, 2016. ACM.

[3] Yun Lin, Xin Peng, Zhenchang Xing, Diwen Zheng, and Wenyun Zhao. Clone-based and interactive recommendation for modifying pasted code. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, pages 520–531, New York, NY, USA, 2015. ACM.

[4] Yun Lin, Jun Sun, Lyly Tran, , and Jinsong Dong. An approach for simulated debugging. 2018.

[5] Yun Lin, Jun Sun, Lyly Tran, Guangdong Bai, Haijun Wang, and Jinsong Dong. Break the dead end of dynamic slicing: Localizing data and control omission bug. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ASE 2018, pages 509–519, New York, NY, USA, 2018. ACM.

[6] Yun Lin, Jun Sun, Yinxing Xue, Yang Liu, and Jinsong Dong. Feedback-based debugging. In *Proceedings of the 39th International Conference on Software Engineering*, ICSE '17, pages 393–403, Piscataway, NJ, USA, 2017. IEEE Press.

[7] Yun Lin, Zhenchang Xing, Xin Peng, Yang Liu, Jun Sun, Wenyun Zhao, and Jinsong Dong. Clonepedia: Summarizing code clones by common syntactic context for software maintenance. In *2014 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, volume 00, pages 341–350, Sept. 2014.

[8] Yun Lin, Zhenchang Xing, Yinxing Xue, Yang Liu, Xin Peng, Jun Sun, and Wenyun Zhao. Detecting differences across multiple instances of code clones. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 164–174, New York, NY, USA, 2014. ACM.

[9] Jun Sun, Yun Lin, Hairui Zhang, Jindong Gao, Xin Peng, and Jinsong Dong. Improving concolic testing through learning. In *IEEE Transaction on Software Engineering (TSE)*, 2018 (under review).

[10] Haijun Wang, Yun Lin, Yang Liu, Zijiang Yang, Jun Sun, Jin Song Dong, Qinghua Zheng, and Ting Liu. Explaining regressions via alignment slicing and mending. In *IEEE Transaction on Software Engineering (TSE)*, 2018 (under review).